

17장. 생성자 함수에 의한 객체 생성

17.1 Object 생성자 함수

1. new 연산자와 함께, Object 생성자 함수 호출 → 빈 객체를 생성하여 반환

2. [예제 17-01]

```
const person = new Object(); // 빈 객체 생성

person.name = "Lee"; // 프로퍼티 추가
person.sayHello = function () {
  console.log('Hi! My name is ' + this.name);
};

console.log(person); // { name: "Lee", sayHello: f }
person.sayHello(); // Hi! My name is Lee
```

3. 생성자 함수(constructor)란,

new 연산자와 함께 호출하여 객체(인스턴스)를 생성하는 함수

4. Object 생성자 함수를 사용해 객체를 생성하는 방식은 특별한 이유가 없다면 그다지 유용 X

17.2 생성자 함수

17.2.1 객체 리터럴에 의한 객체 생성 방식의 문제점

1. 객체 리터럴에 의한 객체 생성 방식은 단 하나의 객체만 생성.

➡ 따라서, 동일한 프로퍼티를 갖는 객체를 여러 개 생성해야하는 경우, **비효율적**

17.2.2 생성자 함수에 의한 객체 생성 방식의 장점

1. 생성자 함수에 의한 객체 생성 방식은, 마치 객체(인스턴스)를 생성하기 위한 템플릿(클래스)처럼

생성자 함수를 사용하여 프로퍼티 구조가 동일한 객체 여러 개를 **간편하게** 생성할 수 있다.

this

- this : 객체 자신의 프로퍼티나 메서드를 참조하기 위한 자기 참조 변수(self-referencing variable)
- this가 가리키는 값, 즉 **this 바인딩**은 함수 호출 방식에 따라 동적으로 결정된다.

2. 생성자 함수는 일반 함수와 동일한 방법으로 생성자 함수를 정의하고, **new 연산자와 함께 호출하면**, 해당 함수는 생성자 함수로 동작한다.

[예제 17-06]

```
const circle3 = Circle(15); // new 연산자가 없어서 일반 함수로서  
  
console.log(circle3); // undefined  
  
console.log(radius); // 15
```

17.2.3 생성자 함수의 인스턴스 생성 과정

1. 생성자 함수의 역할

- 인스턴스를 생성 - **필수**
- 생성된 인스턴스를 초기화(인스턴스 프로퍼티 추가 및 초기값 할당) - **옵션**

2. 인스턴스를 생성하고 반환하는 코드는 보이지 않음.

→ 자바스크립트 엔진은 암묵적인 처리를 통해 인스턴스를 생성하고 반환.

1. 인스턴스 생성과 this 바인딩

a. 암묵적으로 빈 객체 생성

b. 빈 객체, 즉 인스턴스는 this에 바인딩

※ 바인딩 : 식별자와 값을 연결하는 과정을 의미

2. 인스턴스 초기화

this에 바인딩 되어 있는 인스턴스를 초기화한다.

3. 인스턴스 반환

완성된 인스턴스가 바인딩된 this가 암묵적으로 반환된다.

17.2.4 내부 메서드 `[[Call]]`과 `[[Construct]]`

1. 함수는 객체이므로 일반 객체(ordinary object)와 동일하게 동작할 수 있다.

2. 함수는 객체이지만 일반 객체와 다르다.

일반 객체는 호출할 수 없지만 함수는 호출할 수 있다.

3. [예제 17-14]

```
function foo() {}

// 일반적인 함수로서 호출 : [[Call]]이 호출됨.
foo();

// 생성자 함수로서 호출 : [[Construct]]가 호출됨.
new foo();
```


17.2.5 constructor와 non-constructor의 구분

1. 자바스크립트 엔진은 함수 정의를 평가하여 함수 객체를 생성할 때,
함수 정의 방식에 따라 함수를 constructor와 non-constructor로 구분
2. 함수 선언문, 함수 표현식으로 정의된 함수만이 constructor
ES6의 화살표 함수와 메서드 축약 표현으로 정의된 함수는 non-constructor

17.2.6 new 연산자

1. new 연산자와 함께 함수를 호출하면 해당 함수는 생성자 함수로 동작한다.
→ `[[Call]]`이 아닌 `[[Construct]]` 호출
2. 반대로 new 연산자 없이 생성자 함수를 호출하면 일반 함수로 호출된다.
→ `[[Construct]]`가 아니라 `[[Call]]`이 호출
3. 일반 함수, 생성자 함수에 특별한 형식적 차이는 없다.
따라서, 생성자 함수는 **파스칼 케이스**로 명명하여 구별할 수 있도록 한다.
ex) Circle

17.2.7 new target

1. 파스칼 케이스 컨벤션을 사용한다 하더라도 실수는 언제나 발생할 수 있다.
 이러한 위험성을 회피하기 위해 ES6에서는 **new.target**을 지원한다.
2. new.target은 this와 유사하게 constructor인 모든 함수 내부에서 암묵적인 지역 변수와
같이 사용되며 메타 프로퍼티라고 부른다.
3. **new 연산자와 함께 생성자 함수로서 호출되면 함수 내부의 new.target은 함수 자신 가리킴.**

new 연산자 없이 일반 함수로서 호출된 함수 내부의 new.target은 undefined

스코프 세이프 생성자 패턴(scope-safe constructor)

- IE에서는 new.target을 지원하지 않는다. → 스코프 세이프 생성자 패턴을 사용할 수 있다.
4. 대부분 빌트인 생성자 함수(Object, String, Number, Boolean, Date, Array, RegExp 등)은
new 연산자와 함께 호출되었는지를 확인한 후 적절한 값을 반환한다.
- Object, Function 생성자 함수는 new 연산자 없이 호출해도 동일하게 동작.
 - String, Number, Boolean 생성자 함수는 없이 호출하면 문자열, 숫자, 불리언 값을 반환
→ 이를 통해 데이터 타입을 변환하기도 함.